Object-Oriented Software Engineering

Object Orientation

Slide untuk melengkapi buku Applying UML and Patterns
Versi 1.0

Program Studi Teknik Informatika
Jurusan Teknik Informatika
Politeknik Negeri Batam
Jalan Ahmad Yani, Batam Center, Batam 29461
www.polibatam.ac.id

Materi

- Pendahuluan
- Model
- Object Orientation
- Class
- Object
- Encapsulation
- Message
- Inheritance
- Polymorphism
- UML
- Structure Diagram
- Behavior Diagram

Catatan: materi kuliah ini menggunakan istilah-istilah dalam bahasa aslinya yaitu Bahasa Inggris untuk menghindari kesalahan arti

Pendahuluan

- Bahasa pemodelan *(modeling languages)* diciptakan agar orang dari berbagai latar belakang dapat membaca, memahami dan mengimplementasikan berbagai model yang menggambarkan sistem. Bahasa tersebut dapat menggambarkan deskripsi struktur sistem yang akan dibangun.
  - Berbagai bahasa ini dapat berupa tekstual (misal bahasa pemrograman seperti Java) atau visual.

- Pemodelan berorientasi objek (*object-oriented modeling)* adalah sebuah pemodelan yang mengikuti paradigma atau pemahaman berorientasi objek

Model

- Model memungkinkan kita untuk mendeskripsikan sistem secara efisien
  - Sebuah *sistem* adalah kumpulan dari berbagai komponen yang saling terkait satu sama lain dan saling mempengaruhi sehingga kesemua komponen tersebut dapat dianggap sebagai satu unit yang menjalankan satu pekerjaan atau mencapai satu tujuan.
  - Dalam teknologi informasi, kita terutama membahas sistem perangkat lunak dan model-model untuk mendeskripsikan sistem perangkat lunak tersebut.

- Sistem perangkat lunak sendiri berdasar pada *abstractions* yang merepresentasikan fakta yang dapat diproses oleh mesin
  - Sehingga menghasilkan atau melakukan *abstraction* dapat berarti berawal dari yang spesifik kemudian bergerak mencari karakteristik umum yang sama

Model

- Model digunakan sebagai *sketch* untuk mengkomunikasikan berbagai aspek sistem dalam cara yang sederhana
- Model digunakan sebagai *blueprint*, dimana model-model tersebut harus mencakup berbagai detail sehingga developers dapat menghasilkan sistem yang siap diajalnkan tanpa harus membuat keputusan dalam desain
- Model digunakan sebagai *executable programs*, berarti bahwa model dapat dispesifikasi secara tepat sehingga kita dapat menghasilkan kode program secara otomatis

Object Orientation

- Terciptanya konsep *object orientation* dimulai sejak tahun 1960s dengan munculnya bahasa simulasi SIMULA
- Kita melihat dunia nyata sebagai kumpulan yang terdiri dari individu-individu yang dapat berjalan sendiri, disebut sebagai object, yang memiliki peran tersendiri dan harus dapat memenuhi kebutuhan/kewajiban yang telah ditetapkan
- Objects merupakan elemen dalam sisten yang dideskripsikan data dan operasinya. Object saling berinteraksi dan berkomunikasi satu sama lain

Class

- Class mendeksripsikan atribut dan *behavior*/perilaku sekumpulkan objecy (instance dari sebuah *class*) dan mengelompokkan karakteristik objek yang sama

- Sebuah *class* juga mendefinisikan sekumpulan operasi yang dapat diaplikasikan untuk instance dari class tersebut

Object

- Instance dari sebuah *class* disebut sebagai *object*
- Sebuah *object* memiliki identitasnya sendiri, sehingga berbagai instance dari sebuah class dapat dibedakan satu sama lain
- Sebuah *object* selalu memiliki *state*. Sebuah *state* diekspresikan dengan nilai dari *attribute-nya*.
- Sebuah *object* juga menampilkan *behavior*/perilaku. *Behavior* sebuah *object* dideskripsikan dengan himpunan *operation*-nya. *Operation* dapat dijalankan dengan mengirim sebuah *message*.
- Contoh:
  - *R705*, Ruang Kelas 705 Gedung Utama Politeknik Negeri Batam, adalah *object* yang merupakan instance dari class *LectureHall*
  - Infocus di R705 adalah object yang berbeda dari infocus yang ada di R707, walaupun keduanya adalah alat dengan tipe yang sama
  - Sebuah ruang kuliah dapat terisi atau kosong (*state*)

Encapsulation

- *Encapsulation* merupakan perlindungan terhadap akses yang tidak terautorisasi terhadap state internal sebuah onject melalui interface yang sudah ditentukan. Berbagai level visibility dairi interface dapat membantu mendefinisikan akses yang tidak terautorisasi ini.

- Sebagai contoh Java memiliki marker visibility: public, private, dan protected, yang memerikan ijin akses untuk semua, hanya di dalam object, dan hanya untuk anggota dari class yang sama, sub-class-nya, dan package yang sama.

Message

- Object saling berkomunikasi satu sama lain melalui message.
- Sebuah message ke sebuah object merupakan sebuah permintaan untuk menjalankan sebuah operation.
- Object itu sendiri yang menentukan apakah akan menjalankan operation tersebut serta bagaimana caranya.
- Operation hanya dijalankan jika pengirimnya memiliki autorisasi untuk memanggil operation tersebut dan ada implementasi yang tersedia

Inheritance

- Konsep inheritance merupakan sebuah mekanisme untuk membuat class baru dari class yang sudah ada.
- Sebuah subclass diturunkan dari class yang sudah ada (= superclass) mewarisi semua attribute dan operation (baik spesifikasi maupun implementasinya) dari superclass.
- Sebuah subclass dapat:
  - Mendefinisikan attribute dan/atau operation yang baru
  - Mengganti implementasi dari operation yang diwarisi
  - Menambahkan kode ke operation yang diwarisi
- Inheritance menghasilkanhirarki class sebagai dasar pengembangan sistem berorientasi objek. Hirarki class terdiri dari class-class yang memiliki properti yang sama.

Polymorphism

- Polymorphism adalah kemampuan untuk mengadopsi berbagai bentuk.
- Polymorphic operation dapat dijalankan terhadap object dari berbagai class yang berbeda. Hal ini dapat diimplemenetasikan dengan berbagai cara:
    i. Melalui *parametric polymorphism*, lebih dikenal sebagai *genericity*—menggunakan tipe parameter. Sebagai contoh, di Java, concrete class dipindahkan ke operation sebagai salah satu *argument*;
    ii. Melalui *inclusion polymorphism*—operation dapat diaplikasikan ke class dan subclass langsung maupun tidak langsung;
    iii. Melalui *overloading* operation; dan
    iv. Melalui *coercion*, yaitu konversi tipe.

UML

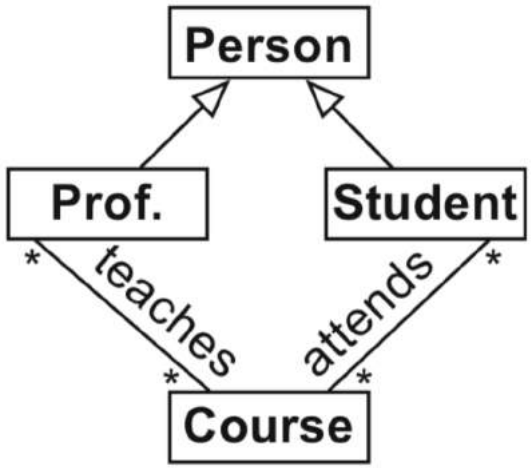
- UML memiliki tujuh jenis diagram untuk pemodelan struktur dari berbagai perspektid yang berbeda. Perilaku/behavior dinamis dari sebuah elemen (atau perubahan dari waktu ke waktu) tidak dimasukkan dalam diagram-diagram tersebut.

Structure Diagram

- Class Diagram
- Object Diagram
- Package Diagram
- Component Diagram
- Composition Structure Diagram
- Deployment Diagram
- Profile Diagram

Class Diagram

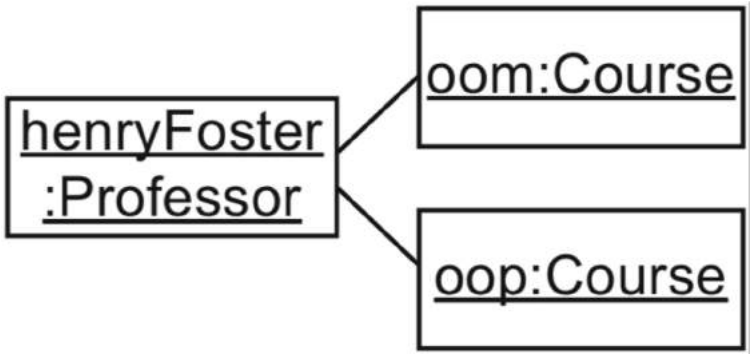
- Digunakan untuk menunjukkan stuktur data dan struktur object di dalam sistem. Class diagram berdasar pada konsep class, generalization, dan association



- Dalam class diagram di samping, kita bisa memodelkan class *Course*, *Student*, and *Professor* yang ada dalam sistem. *Professor* mengajar *course* dan *student* mengikuti *course*. *Student* dan *professor* memiliki property yang sama karena keduanya adalah anggota class *Person*.

Object Diagram

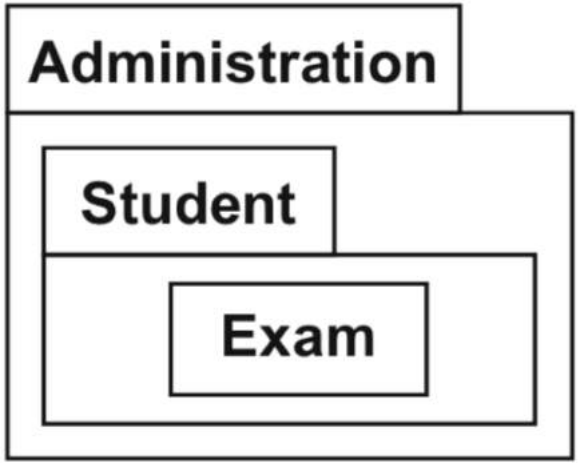
- Object diagram menunjukkan gambaran nyata keadaan sebuah sistem pada suatu saat tertentu.



- Object diagram di samping menunjukkan bahwa seorang *professor* Henry Foster (*henryFoster*) mengajar *course* Object-Oriented Modeling (*oom*) dan Object-Oriented Programming (*oop*).

Package Diagram

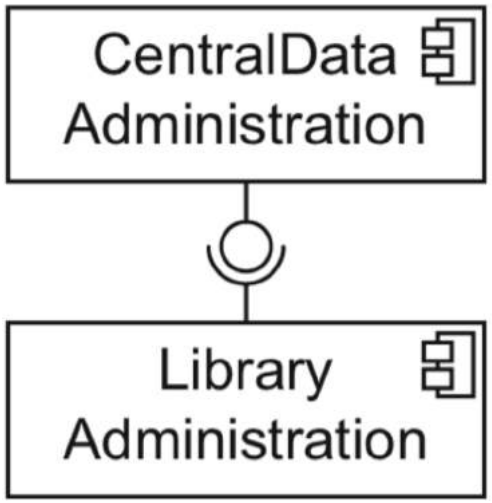
- Package diagram mengelompokkan diagram atau model menurut properti yang sama.
- Package seringkali digabungkan dalam diagram lain daripada digambarkan sebagai diagram terpisah.



- Diagram di samping menunjukkan bahwa dalam sistem administrasi universitas, kita dapat membuat sebuah package yang berisi informasi mengenai aspek pengajaran, riset dan administrastif.

Component Diagram

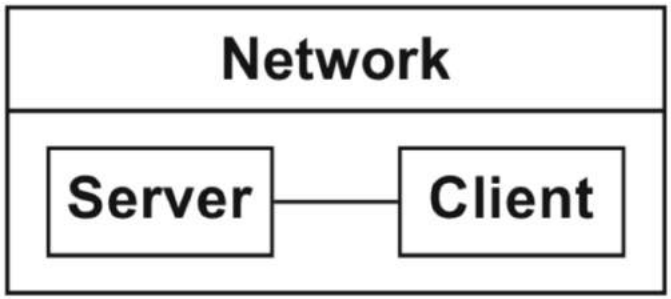
- Component adalah sebuah unit yang independen dan dapat dijalankan untuk memberikan layanan atau service kepada komponen lain atau menggunakan service dari komponen lain.



- Saat mendefinisikan sebuah komponen, kita dapat memodelkan dua pandangan atau view: *external view* (*black box view*), yang menggambarkan spesifikasi komponen, dan *internal view* (*white box view*), yang mendefinisikan implementasi komponen tersebut.

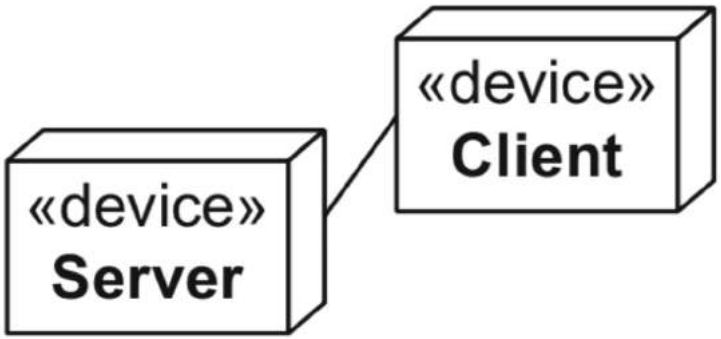
Composition Structure Diagram

- Composition structure diagram menggambarkan hirarki dekomposisi atas bagian-bagian sistem. Sehingga kita dapat menggunakan composition structure diagram untuk mendeksripsikan struktur internal class dan komponen.



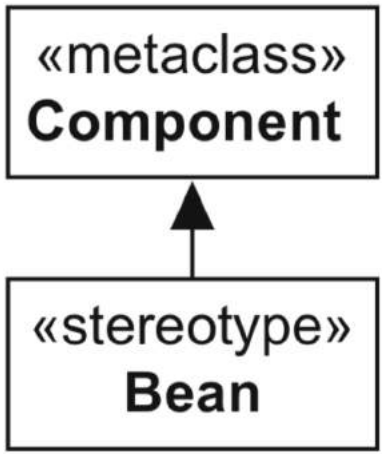
Deployment Diagram

- Topologi hardware dan runtime system yang digunakan dapat digambarkan menggunakan deployment diagram. Hardware menunjukkan processing unit yang digambarkan berupa node dan komunikasi antar nodes. Runtime system berisi artifact yang diberikan ke nodes.



Profile Diagram

- Menggunakan profile, kita bisa memperluas UML untuk memasukkan konsep-konsep yang hanya ada dalam domain tertentu. Definisi UML yang sebenarnya, meta-model, tetap tidak berubah.

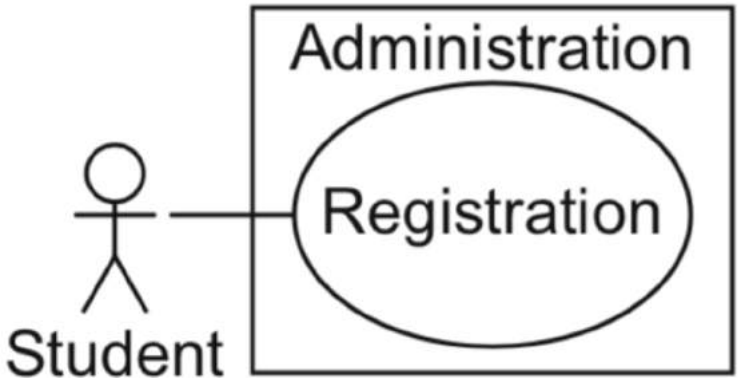


Behavior Diagrams

- Use Case Diagram
- State Machine Diagram
- Activity Diagram
- Sequence Diagram
- Communication Diagram
- Timing Diagram
- Interaction Overview Diagram

Use Case Diagram

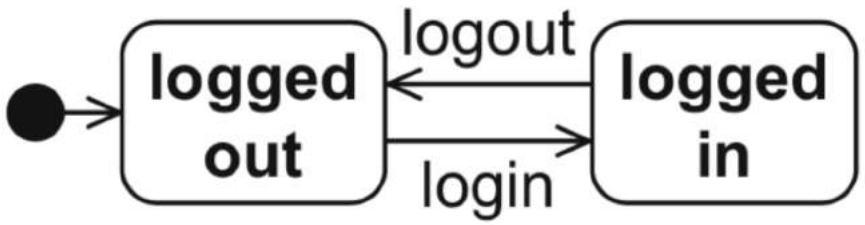
- Use case diagram digunakan untuk mendefinisikan kebutuhan yang harus dipenuhi oleh sistem. Diagram ini mendeskripsikan user mana menggunakan fungsionalitas sistem yang mana, tapi tidak membahas implementasi secara khusus. Sebuah fungsionalitas yang disediakan oleh sistem untuk penggunanya disebut use case.



- Use case diagram di samping memperlihatkan dalam sebuah sistem asministrasi university, fungsionalitas *Registration* merupakan sebuah use case yang digunakan oleh *student*.

State Machine Diagram

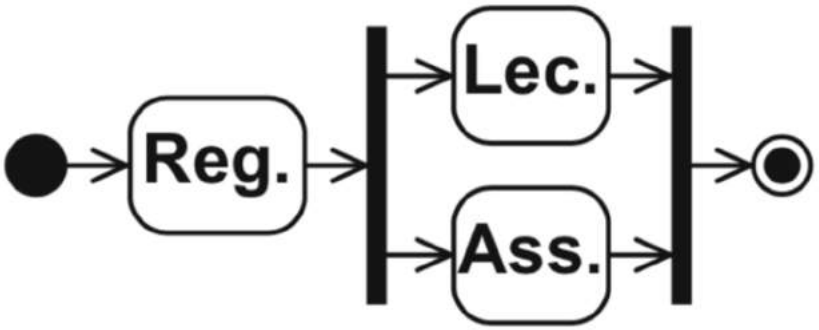
- State machine diagram mendeskripsikan perilaku atau behavior sebuah object yang diperkenankan dalam bentuk *state* dan *state transition* atau perpindahan antar state yang disebabkan oleh berbagai event.
- Dalam masa hidupnya, object akan melalui berbagai state yang berbeda.



- State machine diagram di samping memperlihatkan perubahan state menjadi *logged in* setelah pengguna memasukkan username and password secara benar (event *login*). Begitu pengguna melakukan log out (event *logout*), maka pengguna tersebut kembali ke state *logged out*.

Activity Diagram

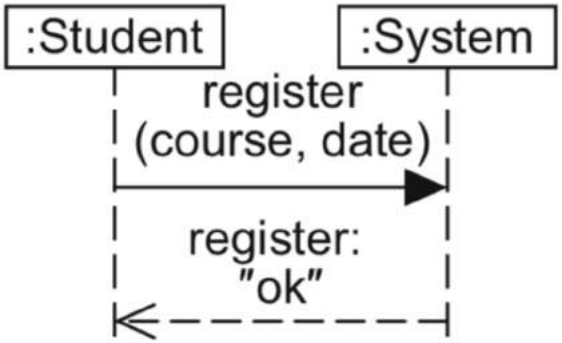
- Kita dapat memodelkan proses apapun menggunakan activity diagram: baik proses bisnis maupun proses software.
- Activity diagram memiliki mekanisme control flow dan mekanisme data flow yang mengkoordinasi aksi yang merupakan bagian dari aktivitas sebuah proses.



- Activity diagram di samping menunjukkan aksi mana yang harus dilakukan oleh student untuk mendaftar lecture dan assignment.

Sequence Diagram

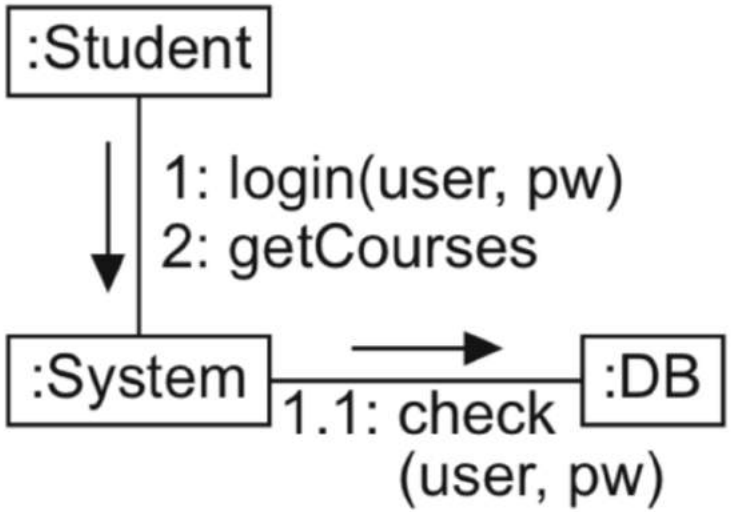
- Sequence diagram mendeskripsikan interaksi antar object untuk menyelesaikan sebuah pekerjaan tertentu.
- Fokusnya adalah urutan kronologis dari pertukaran message atau pesan antar partner yang saling berinteraksi.



- Sequnce diagram di samping memperlihatkan proses registrasi untuk sebuah ujian di dalam sistem adminsitrasi universitas.

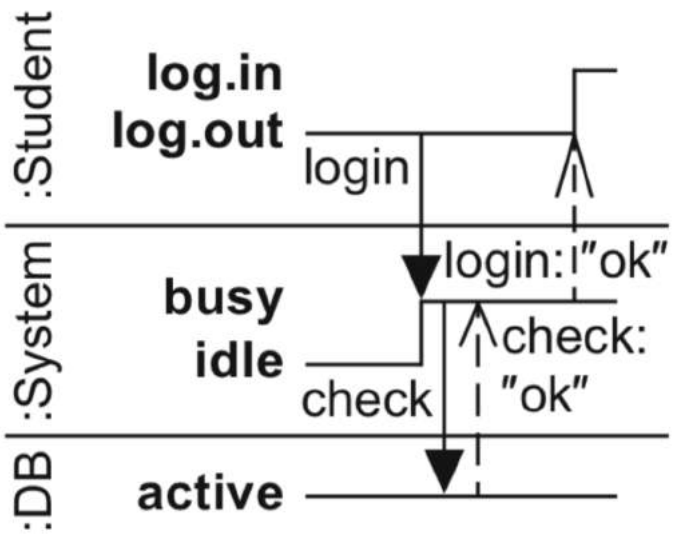
Communication Diagram

- Communication diagram mendeskripsikan komunikasi antara berbagai object. Disini fokusnya adalah hubungan komunikasi antar partner yang saling berinteraksi, bukan pada urutan kronologis pertukaran pesan.
- Diagram ini menunjukkan dengan jelas siapa berinteraksi dengan siapa.



Timing Diagram

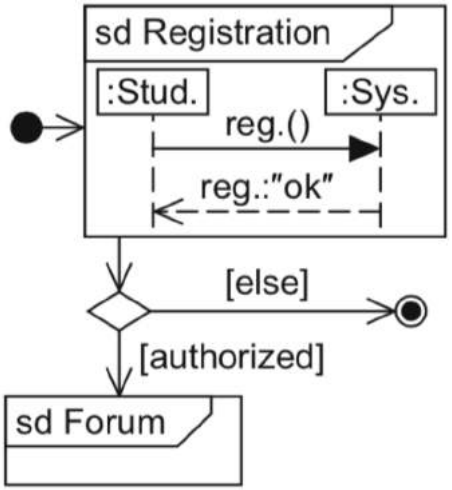
- Timing diagram secara eksplisit menunjukkan perubahan state dari partner yang saling berinteraksi yang dapat terjadi karena adanya time events atau akibat dari pertukaran message.



- Sebagai contoh dalam diagram di samping, seorang pengguna berada dalam state logged begitu message diterima dari sistem administrasi universitas bahwa password yang dikirimkan adalah valid.

Interaction Overview Diagram

- Interaction overview diagram memodelkan hubungan antara berbagai proses interaksi ke dalam interaction diagram masing-masing (yaitu sequence diagram, communication diagram, timing diagram, dan interaction overview diagram lain) dalam urutan berdasarkan waktu atau berdasarkan urutan sebab-akibat. Diagram ini juga menggambarkan kondisi spesifik dimana proses interaksi tersebut diperbolehkan.



- Seorang pengguna sistem adminstrasi universitas harus log in dulu (yang telah memodelkan interaksi dengan sistem) sebelum diperbolehkan menggunakan fungsionalitas lain.